robotics

MDPI

Article

Towards Fully Autonomous Negative Obstacle Traversal via Imitation Learning Based Control

Brian César-Tondreau [1,*], Garrett Warnell [2], Kevin Kochersberger [1] and Nicholas R. Waytowich [2]

[1] Unmanned Systems Laboratory, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA; kbk@vt.edu

[2] Army Research Laboratory, Aberdeen Prooving Ground, Adelphi, MD 21005, USA; garrett.a.warnell.civ@army.mil (G.W.); nicholas.r.waytowich.civ@army.mil (N.R.W.)

* Correspondence: bcesarto@vt.edu

Abstract: Current research in experimental robotics has had a focus on traditional, cost-based, navigation methods. These methods ascribe a value of utility for occupying certain locations in the environment. A path planning algorithm then uses this cost function to compute an optimal path relative to obstacle positions based on proximity, visibility, and work efficiency. However, tuning this function to induce more complex navigation behaviors in the robot is not straightforward. For example, this cost-based scheme tends to be pessimistic when assigning traversal cost to negative obstacles. Its often simpler to ascribe high traversal costs to costmap cells based on elevation. This forces the planning algorithm to plan around uneven terrain rather than exploring techniques that understand if and how to safely traverse through them. In this paper, imitation learning is applied to the task of negative obstacle traversal with Unmanned Ground Vehicles (UGVs). Specifically, this work introduces a novel point cloud-based state representation of the local terrain shape and employs imitation learning to train a reactive motion controller for negative obstacle detection and traversal. This method is compared to a classical motion planner that uses the dynamic window approach (DWA) to assign traversal cost based on the terrain slope local to the robots current pose.

Keywords: autonomous navigation; learning from demonstration; imitation learning

check for updates

Citation: César-Tondreau, B.; Warnell, G.; Kochersberger, K.; Waytowich, N.R. Towards Fully Autonomous Negative Obstacle Traversal via Imitation Learning Based Control. *Robotics* **2022**, *11*, 67. https://doi.org/10.3390/robotics11040067

Academic Editors: Loris Roveda, Braghin Francesco, Sotiris Makris and Petar B. Petrovic

Received: 6 May 2022
Accepted: 20 June 2022
Published: 22 June 2022

1. Introduction

Negative obstacles represent an important challenge in autonomous robot navigation. While structured urban environments with flat, level ground and vertical obstacles can be safely traversed with monocular-camera, stereo-vision, and planar lidar-based navigation, these sensors may not be able to perceive the lower part of negative obstacles such as holes in the ground or downward leading slopes. Therefore the robot has to consider the cause for the missing data from untraversable gaps or simple visibility issues [1,2]. It is worth noting that negative obstacles also present a similar issue for human drivers; a survey conducted by the Government Accountability Office (GAO) [3], found that vehicular "rollovers" represented the most lethal type of accident involving military vehicles, accounting for 63 percent of personnel deaths during non-combat/training scenarios, but only a quarter of surveyed accidents.

Existing approaches to autonomous navigation are largely based on conventional, cost-based path planning techniques which vary with the type of sensing employed in the task. These approaches utilize sensing modalities such as thermal imaging, stereo vision, and 3D lidar [4–6] which are then translated to a location in an internal map where traditional path planning and motion control techniques are used to plan a safe path around it. However, this assumes that there exists a traversable, locally accessible route around the negative obstacle. Meanwhile, researchers have recently tackled a myriad of challenging navigation tasks using machine learning [7–9] to frame the navigation task as an end-to-end learning problem to predict continuous controls directly from raw state observations.

This avoids having to hand specify cost functions and generalized decision-making rules for real-world autonomous navigation. As such, learning behaviors from easy-to-collect human driving demonstrations is an appealing means to create motion controllers that can handle geometrically complex environmental hazards such as negative obstacles.

This paper introduces a behavioral cloning based approach to train an end-to-end policy for traversing through, rather than around, negative obstacles. Specifically, our method allows a lidar equipped UGV to identify and maneuver through slopes that are too steep to be safely traversed without a specific and precise movement protocol. To our knowledge, there is no standard for assessing complex motion control solutions to autonomous traversal problems such as those addressed in this work. As such, an experimental testbed was designed in a simulation comprising four uniquely shaped negative obstacles to develop and evaluate our approach for safe negative obstacle traversal. We trained four separate models to traverse one of the four unique negative obstacle types. We then evaluated each model's ability to successfully navigate through the negative obstacles on which they were trained. We also evaluated their ability to generalize to unfamiliar negative obstacles by successfully completing navigation tasks in the other three environments. Finally, we compared our method's performance to that of the move_base motion planner [10] that we modified to assign traversal cost based on the sensed slope of the terrain in front of to the robot. When applied to the same negative obstacle traversal tasks, we found that the proposed technique excels in performance when compared to move_base.

This paper is organized as follows: Section 2 discusses current autonomous navigation approaches to negative obstacle detection and traversal. In Section 3, we describe all of the facets of the proposed approach, including the state inputs, action space, model architecture, demonstration collection, and training protocols. Section 4 details the experimental setup and simulated environments where we evaluate the performance of the proposed approach. In Section 5, we describe the empirical results of the aforementioned experiments. In Section 6, we discuss the implications of the experiment results, observations, and points where improvements should be made. Finally, we provide closing remarks in Section 7.

2. Related Work

In this section, we discuss how approaches to negative obstacle detection and representation have been used for robot navigation in the literature. Unmanned Ground Vehicles that navigate in unstructured environments are likely to encounter a variety of hazardous terrain features in the form of negative and positive obstacles. Positive obstacles are obstructions to movement that extend 'up' from or exist above a surface such as people, walls, and other vehicles. Negative obstacles typically include cliffs, ditches, and terrain depressions with negative slopes too steep to be safely traversed without the UGV experiencing rollover.

Both positive and negative obstacles can be detected via geometric analysis of range data or object classification from image data, which are then projected onto a terrain map for path planning. However, negative obstacles tend to be more difficult to detect beyond short distances with ranging sensors, as they are occluded by the surface being traversed. Furthermore, they are difficult to classify visually because the visible portion of a negative obstacle may only span a few pixels in the source image. As a result, current research focuses on making negative obstacle detection and classification as robust as positive obstacle detection. However, research rarely addresses the problem of traversing through negative obstacles, as it is typically lumped under the purview of obstacle avoidance problems and are addressed by techniques that plan trajectories around them rather than exploring ways to safely traverse through them.

The negative obstacle detection approach presented by Larson et al. [1] analyzes a 3D lidar scan of the terrain for absences or gaps in the scan, indicating a steep negative slope. They then apply a State Vector Machine (SVM) to classify them initially as *potentially* negative obstacles, while the robot is too far away to perceive the structure of the entire negative obstacle. It then commits to a definite classification when the robot is close enough to be certain.

Hines et al. [11] propose a method of extracting optimistic estimates of the surface shape within the negative obstacle's occluded/unknown space, called *virtual surfaces*. This method produces a heightmap, that labels the unoccupied voxel cells bordering the unknown/unobserved space, as virtual surface voxels. The resulting virtual surface acts as a "best case" imagined foreslope in the unknown space of the negative obstacle, while it is too far away to be directly observed by the sensor. This allows the robot to approach the negative obstacle and path plan through it, as if it had a traversable foreslope, until it is close enough to be proven otherwise. Morton et al., in [12], also address the issues of incomplete information regarding the surrounding terrain due to sensor sparseness and occlusions. They describe a height–length–density (HLD) terrain classifier that generates a 2.5D terrain map from point clouds. Their method uses information propagation between neighboring cells in order to account for missing data in the point cloud scan and subsequently infer the terrain between them.

Chen et al. [6] propose a method that organizes unordered 3D lidar data into an efficient histogram representation to distinguish a traversable flat road plane from negative and positive obstacles. Their approach first projects the unorganized 3D points into an image-like 2D coordinate system called *Lidar-Imagery*. The coordinate axes are realized by the point's yaw and pitch angle in the Lidar's frame of reference, and its single channel pixel value at those coordinates represent the measurement distance of the point. Each row of the Lidar-imagery represents a lidar-scanning line, which is affected differently depending on the type of obstacle it passes through. They then convert the pixel values in the Lidar-imagery into disparity values and represent them in a histogram. Finally, they use the RANSAC algorithm to obtain a linear model of the flat road plane from the histogram, which is subsequently used as a classification rule for distinguishing the road plane from the positive and negative obstacles.

Thermal imaging has also been leveraged for negative obstacle detection. Matthies et al. [4] exploit apparent temperature differences between negative depressions and their surrounding terrain at night to expose negative obstacles as bright spots on an infrared terrain image. This is combined with geometric tests on the terrain to reject or accept the candidate negative obstacle. However, this only addresses the detection of negative obstacles and not their potential for traversability.

3. Approach

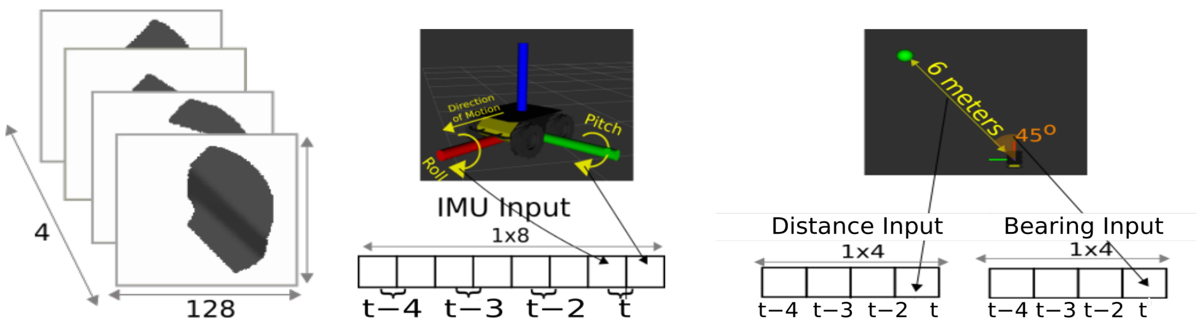
We consider the problem of learning a control policy $\pi_\theta(a_l|s)$, which maps a windowed sequence of state observations $s_{t:t-3} \in S$ to an action class label $a_l \in A$ with the goal of completing short-horizon navigation tasks that involve navigating through negative obstacles. We use behavioral cloning to train this policy from a small structured demonstration dataset of an expert user driving a robotic platform through a negative obstacle without experiencing a rollover.

3.1. State Space

The input to our model consists of the roll and pitch of the robot chassis from an on-board IMU, the point cloud lidar scan of the terrain in front of the robot projected to an 128×128 grayscale image, and task information in the form of the robot's bearing (yaw) and distance to a local goal location on the opposite side of the negative/positive obstacle. We employ a temporal stacking approach (i.e., concatenate data from the time steps immediately preceding the current one) across each of the above mentioned model inputs to provide historical context to the model's predictions.

Surface Normal Input Image: A surface normal input image is constructed using a projection method to convert the 3D point cloud lidar into a 2D grayscale image. We perform this conversion so that we may use deep network architectures designed for 2D images as opposed to those needed to process raw 3D point cloud data directly (e.g., Point-

Net [13] and its iterations), which are conceptually simpler and easier to implement. To generate the image, we first normalize 3D point cloud lidar scans to fit within a unit ball. Next, we compute the surface normal for each point in the normalized point cloud and store each point's surface normal vector in a separate data structure in a way that they can be mapped back to to their corresponding points after the following step. We then project the normalized 3D point cloud to the x–y plane. Afterwards, we generate a new point cloud as an organized $N \times M$ grid of regularly spaced points, where N and M are the desired height and width dimensions for the resulting image, respectively. Finally, for each point in the organized grid, we interpolate the z component of the surface normal vectors of all the normalized point cloud points that fall in a user defined neighborhood of the given grid point. The interpolated values for each grid point are mapped to the corresponding pixels in the grayscale image output (see Figure 1a).



(**a**) Surface Normal Image Input (**b**) IMU Inputs (**c**) Task Based Input

Figure 1. The above images depict the inputs comprising our model's state space detailed in Section 3. (**a**) shows a sequence of grayscale, point cloud images of the terrain in front of the robot observed across four time steps. These are temporally stacked along the channel dimension(depthwise) of a single image. (**b**) shows a sequence of the robot's recorded roll and pitch pairs at each timestep t, over four timesteps. (**c**) shows the sequence of the robots distance and bearing to a given goal location over four timesteps.

IMU Input: This input is the roll and pitch of the robot chassis with respect to the terrain surface. We represent this feature as a 1×2 vector. It is meant to help the model distinguish between traversal on flat terrain from traversal on the incline surfaces (see Figure 1b).

Task–Specific Inputs: We represent the task–specific inputs in two separate vectors. The bearing input vector is the yaw offset between the robot and the goal location at time t. The distance feature vector is the Euclidean distance between the robot and the goal location at time t. These are meant to give the robot a target location on the opposite side of the negative obstacle to traverse (see Figure 1c).

3.2. Windowed State Observation

We employ a temporal stacking technique for each of the features described in Section 3.1 to prevent state aliasing by supplying the learner with historical context for the predicted actions at each time step. We stack a sequence of four observations of the inputs mentioned above: what was just observed in the current time step, followed by the observations made in the three previous time steps. This amounts to the surface normal input image being stacked along the channel dimension, resulting in a $128 \times 128 \times 4$ image from four single channel grayscale images of the same width and height dimensions, as shown in Figure 2. We stack four 1×2 vectors, containing the roll and pitch of the robot chassis, with respect to the terrain surface, into a 1×8 row vector. Finally, we construct two 1×4 vectors from four scalar measurements of the robot's distance and bearing to the goal location. In order to keep these shapes, the temporal stacks are initialized with zero vectors of the same dimensions as our model inputs. The state observed at start time is pushed into the bottom of the stack and the oldest is automatically removed from the top.

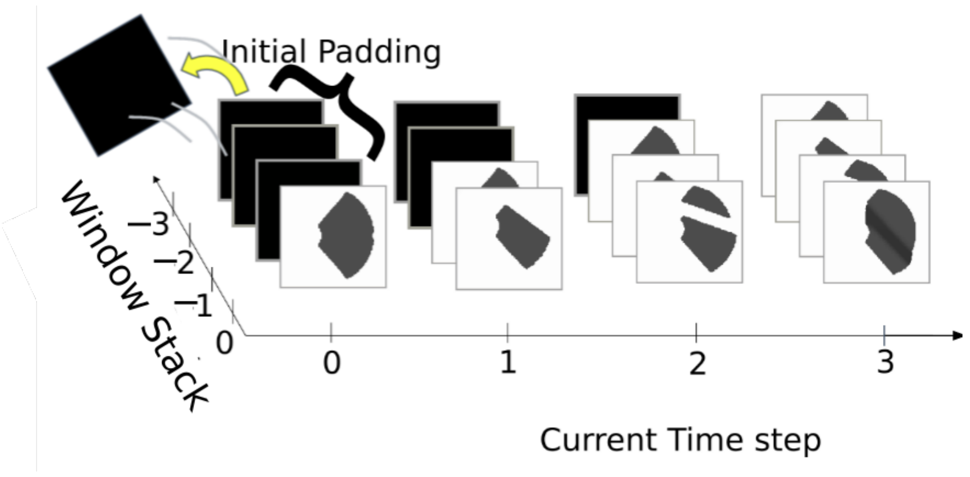
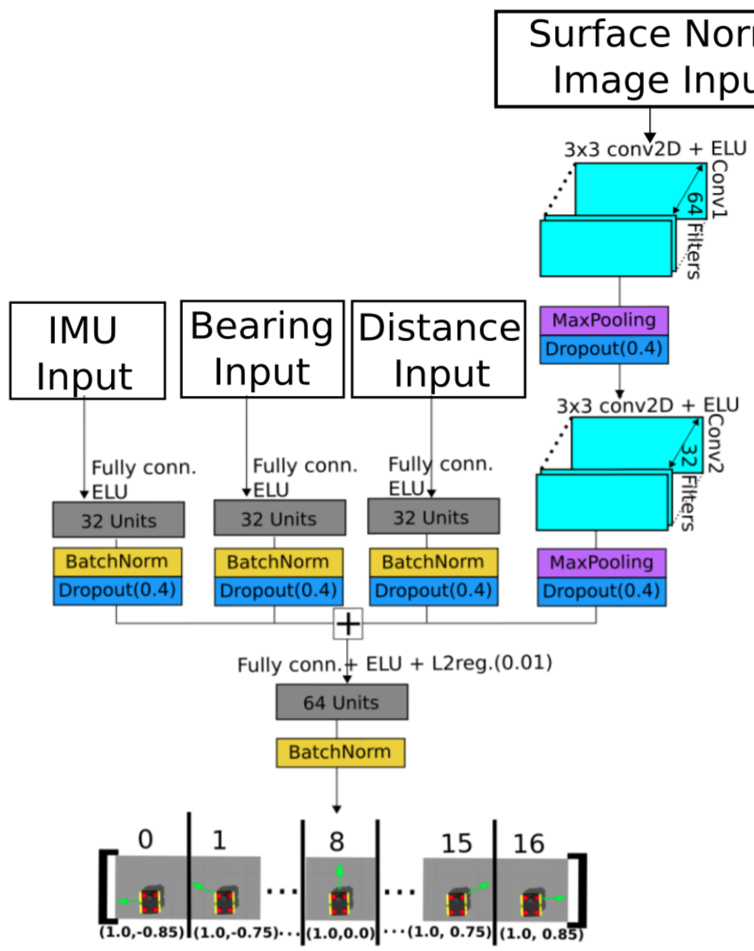


Figure 2. Temporal stacking for the surface normal input image.

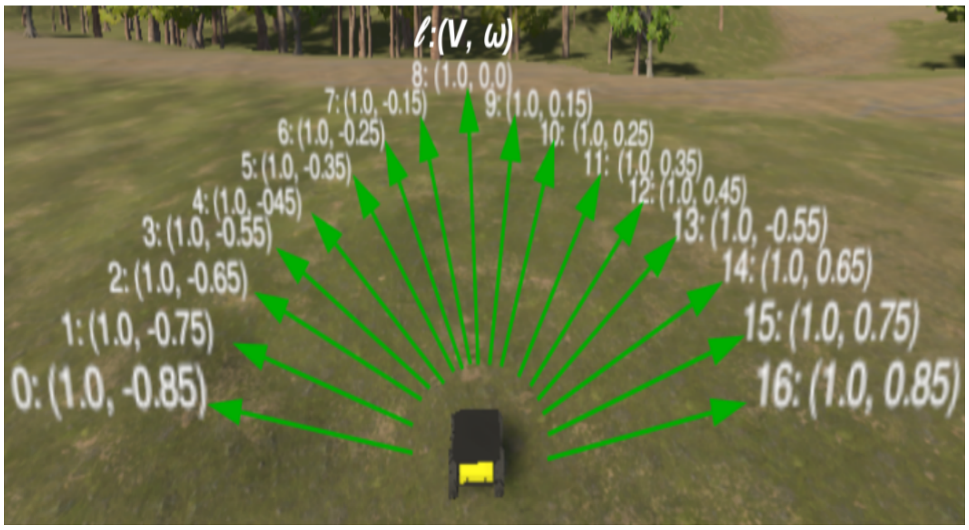
3.3. Action Space

Our action space is composed of seventeen translational v and angular ω velocity command pairs, $a_l \in \{(v,\omega)|v = 1.0, -0.85 \leq \omega \leq 0.85\}$. Each pair is uniquely assigned a well defined and discrete integer class label from 0 to 16, $A = \{a_l|l = 0, 1, \ldots, 16\}$ as shown in Figure 3b, which acts as the target class that our model tries to predict during training and execution. In this work, only the angular velocity commands vary across all labels, as they represent the turning rate that the robot needs to reach the requisite angle of approach. They take on values between $[-0.85, 0.85]$ rads/sec sampled at regular intervals of 0.1 rads/s. Each corresponding translational velocity command is held at 1.0 m/s. We determined experimentally that these 17 movement commands represented an adequate action space size for the proposed approach. A total any higher than this would not improve performance via increased turning precision, while a smaller action space may not be precise enough for the UGV to accurately mimic the demonstrated behavior. Additionally, there is an upper limit to the precision of the movements that the human user can demonstrate comfortably via teleoperation. When moving through rough, uneven terrain, for example, the difference between a 3 and 6 degree turn will most likely be imperceptible to the demonstrator and difficult for the UGV to reproduce.

The reasons for varying the angular velocity and not the translational velocities in our action space were as follows: It was determined experimentally that the angle of approach and maintenance of this angle, prior to and during the agent's traversal through the foreslope and backslope of the negative obstacle, was the key factor to its successful traversal. Furthermore, the human demonstrators never varied the translational velocity of the robot during demonstration, but they did vary the translational velocity, indicating that a maximum translational speed of 1.0 m/s was sufficient to successfully complete the given traversal task.



(**a**) Network Architecture



(**b**) Action space

Figure 3. (**a**) depicts the imitation learning module that maps the state observations, described in Section 1, to one of seventeen class labels that constitutes the action space. (**b**) is a visual depiction of the action space. Each class label, paired with their respective translational and angular velocity command pairs (in white), is pointed to by green arrows which correspond to the direction that the robot will move.

3.4. Network Architecture

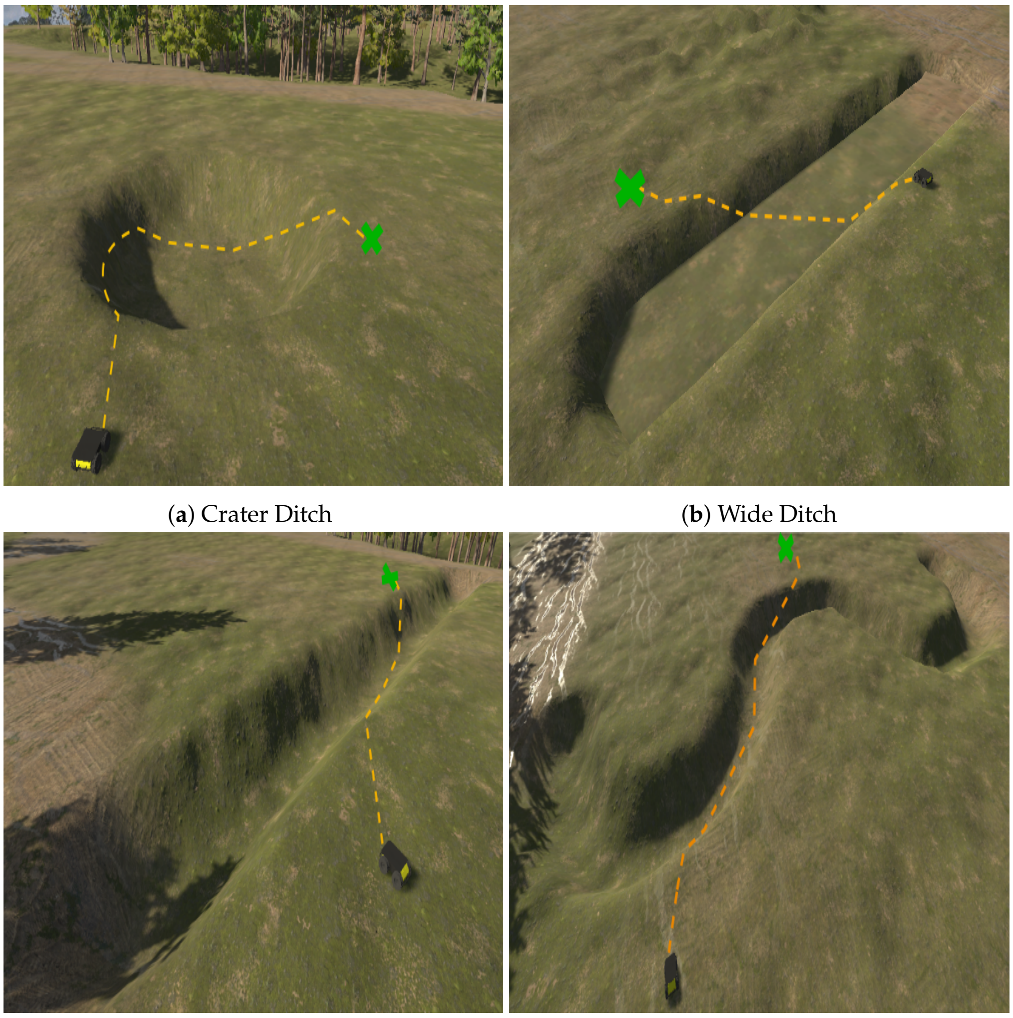
We trained a classification deep network that takes in four separate inputs: the IMU inputs, the surface normal input image, and the two task specific inputs detailed in Section 3.1. Our network architecture for the Behavioral Clone is shown in Figure 3a. The point cloud image inputs move through two 2D convolutional layers, the first and second having 64 and 32 hidden units, respectively. Both layers have a 3×3 kernel size and ELU activation. Each of the resulting feature maps are passed through a max pooling operation, followed by a dropout regularization of 40%, and finally flattened to a vector shape. The IMU, Distance, and Bearing to goal inputs are each passed through separate, fully connected layers. All three layers have 32 hidden units and ELU activation. They are each followed by a batch normalization and dropout regularization of 40%. The feature vectors resulting from the above operations are concatenated and passed through a fully connected layer with 64 hidden units, ELU activation, and batch normalization before finally being passed through a fully connected output layer with SoftMax activation, and seventeen hidden units matching the discrete action labels comprising our action space. Finally, the model is optimized using a categorical cross entropy loss. We chose a classifier model rather than a regressor model, because training the latter would require more training data to account for the real valued velocity commands recorded during training.

3.5. Demonstration Data Collection

In this section, we describe our general procedure for demonstration collection in addition to discussing the issue of data imbalance, common when recording driving data, and the steps we took to address it.

To obtain our training data set, the expert user teleoperates a simulated Husky UGV, using a DualShock PS4 controller, from an initial pose on one side of the negative obstacle to a terminal pose on the opposite side (see Figure 4a). The objective is to travel through the negative obstacle in a way that does not cause the robot to roll over. At each time step during demonstration, the raw velocity commands issued by the expert user are converted to the most similar labeled velocity command a_l comprising our action space and then executed in place of the raw command. Its corresponding class label is recorded along with the state s_t observed at that timestep. We do this to maintain consistency between the robot's movements during demonstration and its movements while under the control of the learned model.

Our demonstration procedure, as well as the nature of the task being demonstrated, is highly likely to generate an imbalanced data set. Driving data are inherently heavily imbalanced, where most of the captured data will be of the vehicle driving straight. During demonstration collection, the critical behaviors for safe traversal through the negative obstacle (i.e., when the UGV turns to adjust its angle of approach to the edge of the slope) occurred at far fewer points in the demonstration than the behaviors for forward motion. As a result, the class label $a_8 = (1.0, 0.0)$ corresponding to pure forward translational velocity was grossly over represented the distribution of training examples. Training with this imbalanced dataset resulted in a model that would over-fit to the move forward command and rarely predict any other ones during execution. We addressed this data imbalance problem by employing the Synthetic Minority Over-sampling Technique (SMOTE) [14] on our training data set. The SMOTE aggregates the training dataset with synthetically generated instances of the minority class examples.


(**a**) Crater Ditch (**b**) Wide Ditch

(**c**) Narrow Ditch (**d**) Sine Ditch

Figure 4. Figures (**a**–**d**) show the environments where both demonstration collection and model evaluation are carried out. The green "x" marks the robot's goal location for a navigation task. The yellow dashed line connecting the green "x" to the robot is the trajectory demonstrated by the expert user to train the model.

3.6. Training Protocol

In this section, we describe our training procedure. We will discuss issues that our method encountered with regards to poor validation accuracy and the resulting model's inattention to specific inputs, as well as the procedures we implemented to fix them.

Validation Accuracy: Our action space comprises seventeen class labels, each aliasing a translational and rotational velocity control signal pair. As each label serves as a discrete placeholder for continuous control signals, there is an acceptable margin of error when comparing the predicted class label with the ground truth that was not accounted for in the accuracy metric during training. For example, the class label $a_{11} = (1.0, 0.35)$, can be safely "misclassified" as either label $a_{12} = (1.0, 0.45)$ or $a_{10} = (1.0, 0.25)$ as, for our purposes, the difference in their respective rotational velocity control signals are negligible. However, misclassifying label a_{12} for a_{10} or some other distant class label would diminish the performance of the resulting model. Our solution was to create an ordinal accuracy metric that accounts for the fact that the label our model predicts need not match the ground truth label exactly when evaluating the models accuracy on a validation data set. The ordinal accuracy metric simply has a settable parameter for acceptable classification tolerance between what the model predicts and the ground truth label. In our application, we found that a classification tolerance of ±1 was sufficient to improve validation accuracy and train a generalizable model.

Input Inattention: Finally, we found that the model seemed to learn solely from the point cloud image inputs during training and ignored the IMU and task specific inputs entirely. At test time, the model would only move forward (not turning to orient itself with the goal as we expected) until it observed a change in terrain elevation from the surface normal input image. While traversing the foreslope and the backslope of the negative obstacle, it made no adjustments to its trajectory to keep the robot at the demonstrated roll and pitch to prevent rolling over. We hypothesized that while one learning rate worked well for the convolutional layers processing the point cloud image input, it may have had the gradient bouncing around the local minimum for the dense layers processing the lower dimensional inputs. To resolve this issue, we implemented layer specific learning rates in our network. For the purposes of our work, we reduced the learning rate for the scalar inputs by a factor of ten with respect to the learning rate of the point cloud image inputs to promote search of a local minimum for the scalar inputs. This kept the step size from being so big that the search for the global minimum does not bounce around the minimum.

4. Experiment

We perform simulation experiments to evaluate the performance of our approach in terms of navigation success. Furthermore, we compare our trained model's performance to an off-the-shelf navigation stack using a hand specified costmap for negative obstacle traversal.

4.1. Simulation Environment

This work aims to develop a learning-based approach for learning specific motion control behaviors that safely traverse negative obstacles without rolling over. Most works in this field examine topics in obstacle detection, assessment of terrain type, and control strategies for collision avoidance. However, they rarely delve deeper into task complexity that requires analysis of terrain geometry and development of control strategies more complex than navigating around the perceived obstacle. As a result, there are a plethora of simulation environments to test various obstacle avoidance problems in vision-based [15,16], socially aware [17], and dynamic [18] contexts. However, there is no standard set of simulation environments for assessing more complex motion control strategies. To this end, we designed four environments, simulated in the Unity game engine, each with a uniquely shaped negative obstacle for the development and assessment of negative obstacle traversal control strategies. Our primary design consideration was to ensure that the negative obstacle backslopes and foreslopes were steep enough to make traversal non-trivial without the correct movement strategy. Specifically, to have a smooth transition from the hinge point of the ditch to the foreslope, the UGV would need to approach the hinge point at a shallow angle. It would then need to maintain that angle as it drives down the foreslope until it reaches the bottom of the ditch, then finally repeating this same strategy to safely travel up the backslope to the opposite side. A trivial motion strategy would have the robot approach the hinge point at a perpendicular angle, making the transition to the foreslope abrupt enough to cause the back wheels lose contact with the ground, tipping the chassis forward over the front bumper and causing rollover. In the case of traversing up the backslope, the trivial motion strategy would result in the front two wheels losing contact with the ground, forcing the robot's back wheels to continue forward underneath the robot until they pass its center mass, again causing a rollover.

4.2. Simulation Setup

The demonstration collection and evaluative simulations were performed using a $0.990 \times 0.670 \times 0.39$ m simulated Clearpath Husky. This Husky is a ground vehicle with 4-wheel differential drive kinematics and a lidar with 120° horizontal field of view, 80° vertical field of view, and a four meter effective sensing range. All simulations and demonstrations were conducted in simulated open field environment created using the Unity simulator. The negative obstacles used are $30 \times 4 \times 4$ m ditches with flat bottoms and walls sloped at approximately 55°. We chose this slope angle to simulate an incline that is traversable only if approached at a shallow angle with respect to the slope edge. The training dataset is composed of forty total demonstrations of the human expert completing three different traversal tasks in each of the four environments depicted in Figure 4.

We assess the models trained using our proposed approach within two simulation contexts: scalability and generalization. In the scalability experiments, we test the effect that the number of demonstration data used to train each model has on success rate. Within each environment, we trained three models with ten, twenty, or forty demonstrations of the robot navigating between three start and end location pairs on either side of the negative obstacle. We then rollout each of the three models to navigate between three different starting and goal location pairs, ten times each, in their respective training environments. At the end of each rollout, the amount of times that the policy successfully navigated to the goal location without experiencing rollover was recorded.

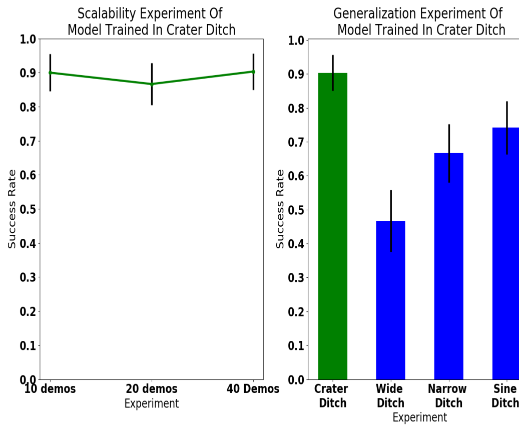
In the generalization experiments, we test each model's ability to generalize to similar navigation tasks in novel environments from where they were originally trained. We evaluated four models trained with forty demonstrations in their respective training environments in the three remaining environments. Each model was tasked with navigating between three different starting and goal location pairs, ten times each, in each of the novel environments, and we noted the amount of times that the policy successfully navigated to the goal location without experiencing rollover.

This approach was compared to the move_base motion planning software used for the Robotics Operating System (ROS) navigation stack, where the move_base navigator used the dynamic window approach (DWA) [19] for local planning and Dijkstra's algorithm for global planning. Originally, the global and local 2D costmaps, used by move_base, assigned a specific value of cost to each cell that was inversely proportional to the robot's distance form a sensed obstacle. This layer was adapted to derive cost from the z-component of the surface normal of each point cloud point in the world reference frame. Specifically, the traversal cost through a region of the terrain increases with increasing or decreasing terrain slope, while flat terrain garners zero cost.

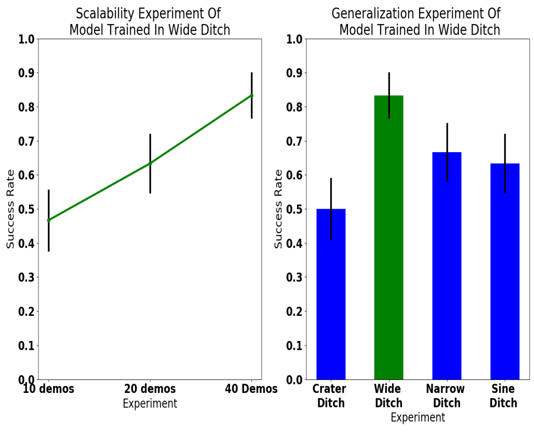
5. Results

The methods for negative obstacle perception and traversal introduced in this paper produced results that are discussed below and illustrated in Figure 5. They indicate that, given a minimal amount of demonstration data, the proposed method can generate trajectories that can successfully traverse various negative obstacles, with significantly higher completion rates than the industry standard navigation software.

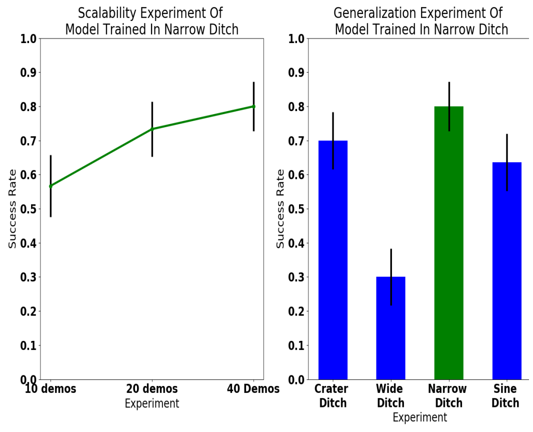
For each evaluation task, the number of trials that the model successfully navigated from the starting pose to the goal pose without experiencing rollover (i.e., the success rate) was recorded. The results of the scalability experiment, depicted as line plots in Figure 5, show that the performance of each policy generally increases with the amount of demonstrations used to train it. The only exception is the policies trained in the Crater environment, which maintain a consistently high success rate across all training demonstration amounts. Furthermore, we observed that the Sine and Narrow Ditch policies start to level off in performance between 70% and 80% success after training with a dataset size of twenty demonstrations. The Wide Ditch policies strictly increase with greater demonstrations used during training. From this, one may conclude that the type of training environment has an effect on the maximum performance one can expect from this approach.



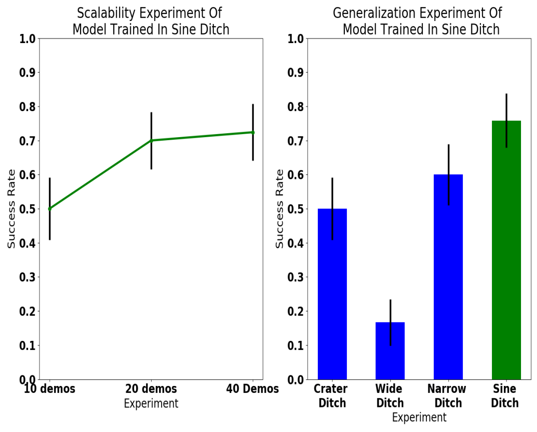
(**a**) Crater Ditch Trained Model Navigation Success Rate



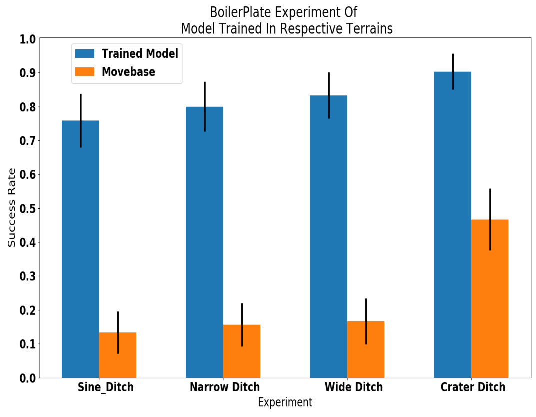
(**b**) Wide Ditch Trained Model Navigation Success Rate



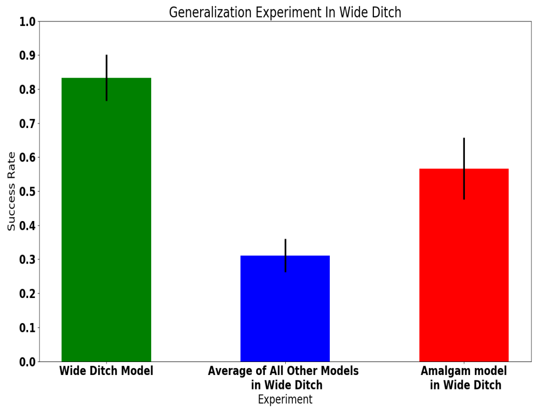
(**c**) Narrow Ditch Trained Model Navigation Success Rate



(**d**) Sine Ditch Trained Model Navigation Success Rate



(**e**) move_base vs. Trained model Success Rates



(**f**) Generalization Experiment in Wide Ditch Success Rate

Figure 5. (**a**–**d**) show the navigation success rates for each of the four models trained, the Crater, Wide, Narrow, and Sine ditch environments, respectively, under two evaluative contexts. The line graph on the left of each figure shows the success rate of the model trained with increasing amounts of demonstration data and evaluated in its training environment. The bar graph on the right shows the success rates of the model that was trained with the maximum number of demonstrations and evaluated in the three other environments. (**e**) is a bar graph comparing the success rates of the each trained model executed in their respective training environments to that of move_base. (**f**) shows a case study of generalization rollouts conducted in the Wide Ditch environment.

The bar plots in Figure 5 summarize the ability for each policy to generalize to negative obstacles that are dissimilar to the ones in their respective training environments. Evidently, the policy trained in the Crater environment was the best at generalizing to novel environments, followed closely by the Wide Ditch policy, then the Narrow Ditch policy, and then, worst by far, the Sine Ditch policy.

Figure 5e compares the performances of the models trained using our approach to the move_base navigation software. The policies trained using our approach all performed significantly better at navigating through their respective training environments than move_base. One can see that move_base's performance increases with decreasing terrain complexity. It was observed that, during execution, move_base would often halt the robot's movement to replan at very inopportune times along its previous path. This typically occurred while the robot was traversing the backslope of the negative obstacle, causing the robot slide down the backslope then roll over towards the base of the ditch. Finally, Figure 5f shows a case study of generalization rollouts conducted in the wide ditch environment. The green bar shows the success rate of Wide Ditch trained model when rolled out in its training environment, the blue bar is the average success rate of Crater ditch, Narrow Ditch, and Sine Ditch trained models when rolled out in the Wide Ditch environment, and the red bar is the success rate of a model trained with an amalgam of demonstrations from those same three environments that was rolled out in the Wide Ditch environment. As depicted in Figure 5a,c,d, their respective trained models individually performed poorest when tasked to traverse through the Wide Ditch environment. We conducted an experiment to see if a model trained from an amalgam of demonstrations from all three of these scenarios would achieve better generalization in the environment where the individual models trained from these demonstrations performed the worst. Figure 5f compares the success rate of the amalgam model to the average success rates of the Crater, Narrow, and Sine Ditch trained models to show that demonstration data from multiple environments that are dissimilar to the target environment will still improve generalization. From this, we could extrapolate that repeating this experiment for the other environments would show similar results.

6. Discussion

Our experimental results illustrate the premise of our proposed method for using Imitation Learning to safely traverse trough negative obstacles. Our approach was able to learn how to safely traverse through one type of negative obstacle then generalize that behavior to safely traverse new ones with very little demonstration data. During the generalization experiments, we noted that the policy trained in the Crater Ditch environment was able to generalize to novel environments better than the other three. This result was contrary to our expectations, because the geometric simplicity of the Crater Ditch environment resulted in very uniform demonstrations of the robot traversing through it. As such, we expected the policy trained from this data to over fit to this environment. In fact, we expected the policy trained in the Sine Ditch environment to have the greatest performance as the geometric complexity of its training environment provided a wider variety of experiential data than the others. While the presented results do establish the efficacy of this approach relative to the baseline, there were certain limitations that are addressed further in this section. One driving assumption of this work is that the negative obstacles encountered would be traversable given the appropriate navigation behavior. As a result, there was very little variation in the incline of our slopes, which may not always reflect reality and may present potential issues of this approach's robustness in real world applications. As such, this approach could perhaps benefit from introducing searching behaviors for finding viable points of ingress and egress by identifying slopes that are too steep for the physical platform to safely traverse. It was also observed that the trained policy would occasionally fluctuate between two opposing actions (i.e., predicting a slight left turn then a slight right turn afterwards, repeatedly) when it encountered unfamiliar states, often resulting in a rollover. This issue could be mitigated by implementing a shared control approach whereby control is arrested from the policy when prediction uncertainty is sufficiently high, and then hands it to a human expert to provide "tuning" demonstrations in the middle of policy rollout.

7. Conclusions

This paper presented a novel method for robotic agents to quickly learn how to traverse negative obstacles from expert human demonstration. Specifically, this work introduced a novel point cloud based state representation of the local terrain shape and used imitation learning to train a reactive motion controller for negative obstacle detection and traversal, instead of modular path planning and control. Furthermore, to our knowledge, there are no standard testing environments for evaluating the performance of new methods that dress the problem of negative obstacle traversal. To this end, four simulation environments were developed to study this problem and test the how precisely our model can learn a human demonstrated technique for safely traversing negative obstacles. It was shown, in simulation, that our proposed method could imitate complex navigation behaviors and generalize to novel environments with minimal demonstration. Additionally, the proposed method was compared to a classical motion planning algorithm which was modified to assign traversal cost based on the terrain slope local to the robots current pose.

Author Contributions: B.C.-T.: conceptualization, methodology, data acquisition, data analysis, validation, and manuscript writing. K.K., G.W. and N.R.W.: supervision and manuscript writing. All authors have read and agreed to the published version of the manuscript.

Funding: The research is supported by ORAU.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Videos of the simulations discussed the results section of this article are openly available and can be found at: https://kairoslaboratory.github.io/traversing-negative-obstacles/ (accessed on 13 June 2022).

Acknowledgments: The authors would like to thank Virginia Polytechnic Institute and State University and the ARMY Research Lab (ARL) for providing all facilities during this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Larson, J.; Trivedi, M. Lidar based off-road negative obstacle detection and analysis. In Proceedings of the 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), Washington, DC, USA, 5–7 October 2011; pp. 192–197.
2. Sinha, A.; Papadakis, P. Mind the gap: Detection and traversability analysis of terrain gaps using LIDAR for safe robot navigation. *Robotica* **2013**, *31*, 1085–1101. [CrossRef]
3. Russell, C. Military Vehicles: Army and Marine Corps Should Take Additional Actions to Mitigate and Prevent Training Accidents. Available online: https://www.gao.gov/products/gao-21-361 (accessed on 19 June 2022).
4. Matthies, L.; Rankin, A. Negative obstacle detection by thermal signature. In Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453), Las Vegas, NV, USA, 27–31 October 2003; Volume 1, pp. 906–913.
5. Hu, T.; Nie, Y.; Wu, T.; He, H. Negative obstacle detection from image sequences. In Proceedings of the Third International Conference on Digital Image Processing (ICDIP 2011). International Society for Optics and Photonics, Chengdu, China, 15–17 April 2011; Volume 8009, p. 80090Y.
6. Chen, L.; Yang, J.; Kong, H. Lidar-histogram for fast road and obstacle detection. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 1343–1348.
7. Chiang, H.T.L.; Faust, A.; Fiser, M.; Francis, A. Learning navigation behaviors end-to-end with autorl. *IEEE Robot. Autom. Lett.* **2019**, *4*, 2007–2014. [CrossRef]
8. Hawke, J.; Shen, R.; Gurau, C.; Sharma, S.; Reda, D.; Nikolov, N.; Mazur, P.; Micklethwaite, S.; Griffiths, N.; Shah, A.; et al. Urban driving with conditional imitation learning. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 31 May–31 August 2020; pp. 251–257.
9. Pan, Y.; Cheng, C.; Saigol, K.; Lee, K.; Yan, X.; Theodorou, E.A.; Boots, B. Agile Off-Road Autonomous Driving Using End-to-End Deep Imitation Learning. *arXiv* **2017**, arXiv:1709.07174. Available online: http://xxx.lanl.gov/abs/1709.07174 (accessed on 8 April 2022).

10. Marder-Eppstein, E. Move_Base—Ros Wiki. 2016. Available online: http://wiki.ros.org/move_base (accessed on 8 April 2022) .
11. Hines, T.; Stepanas, K.; Talbot, F.; Sa, I.; Lewis, J.; Hernández, E.; Kottege, N.; Hudson, N. Virtual Surfaces and Attitude Aware Planning and Behaviours for Negative Obstacle Navigation. *arXiv* **2020**, arXiv:2010.16018. Available online: http://xxx.lanl.gov/abs/2010.16018 (accessed on 23 March 2022).
12. Morton, R.D.; Olson, E. Positive and negative obstacle detection using the HLD classifier. In Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 25–30 September 2011; pp. 1579–1584.
13. Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv* **2016**, arXiv:1612.00593. Available online: http://xxx.lanl.gov/abs/1612.00593 (accessed on 8 September 2021).
14. Bowyer, K.W.; Chawla, N.V.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic Minority Over-sampling Technique. *arXiv* **2011**, arXiv:1106.1813. Available online: http://xxx.lanl.gov/abs/1106.1813 (accessed on 5 November 2021).
15. Hrabar, S. 3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs. In Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, 22–26 September 2008; pp. 807–814. [CrossRef]
16. Cho, G.; Kim, J.; Oh, H. Vision-Based Obstacle Avoidance Strategies for MAVs Using Optical Flows in 3-D Textured Environments. *Sensors* **2019**, *19*, 2523. [CrossRef] [PubMed]
17. Smith, T.; Chen, Y.; Hewitt, N.; Hu, B.; Gu, Y. Socially aware robot obstacle avoidance considering human intention and preferences. *Int. J. Soc. Robot.* **2021**, 1–18. [CrossRef] [PubMed]
18. Lee, J.; Grey, M.X.; Ha, S.; Kunz, T.; Jain, S.; Ye, Y.; Srinivasa, S.S.; Stilman, M.; Liu, C.K. Dart: Dynamic animation and robotics toolkit. *J. Open Source Softw.* **2018**, *3*, 500. [CrossRef]
19. Fox, D.; Burgard, W.; Thrun, S. The dynamic window approach to collision avoidance. *IEEE Robot. Autom. Mag.* **1997**, *4*, 23–33. [CrossRef]